



CSCE 421: Machine Learning

Lecture 9

Overview

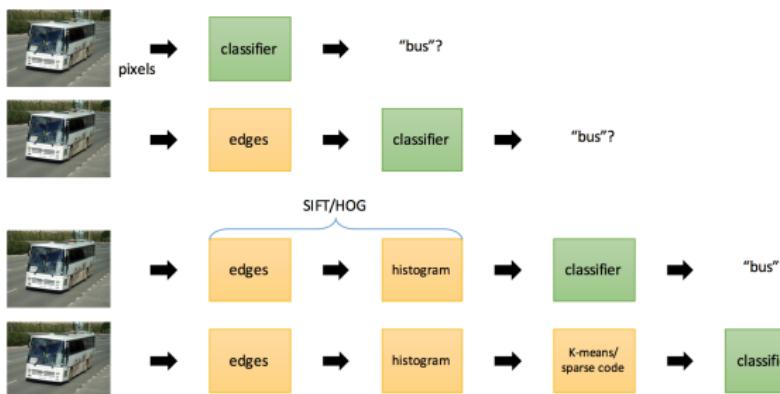
- Deep neural networks
 - Motivation & Challenges
 - Unsupervised pretraining: Deep belief networks & autoencoders
 - (Supervised) fine-tuning
 - Convolutional neural networks

Overview

- Deep neural networks
 - Motivation & Challenges
 - Unsupervised pretraining: Deep belief networks & autoencoders
 - (Supervised) fine-tuning
 - Convolutional neural networks

Deep neural networks: Motivation

Traditional recognition



But what's next?



Deep neural networks: Motivation

Deep Learning

Specialized components, domain knowledge required



Generic components ("layers"), less domain knowledge



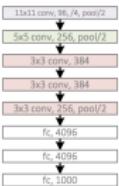
Repeat elementary layers => Going deeper



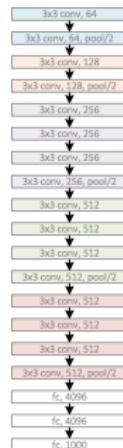
- End-to-end learning
- Richer solution space

Deep neural networks: Motivation

AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



GoogleNet, 22 layers
(ILSVRC 2014)



Deep neural networks: Motivation

Why go deep?

- Deep Representations might allow for a hierarchy or representation
 - Non-local generalization
 - Comprehensibility
- Multiple levels of latent variables allow combinatorial sharing of statistical strength
- Deep architectures work well (vision, audio, NLP, etc.)!

[Contents for the following slides have been summarized from the NIPS 2010, CVPR 2012, and Stanford Deep Learning Tutorials]

Deep neural networks: Motivation

Key ideas of (deep) neural networks

- Learn features from data
- Use differentiable functions that produce features efficiently
- End-to-end learning: no distinction between feature extractor and classifier
- “Deep” architectures: cascade of simpler non-linear modules

Deep neural networks: Motivation

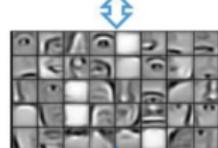
Different levels of abstraction:Hierarchical learning

- Natural progression from low level to high level structure as seen in natural complexity
- Easier to monitor what is being learnt and to guide the machine to better subspaces
- A good lower level representation can be used for many distinct tasks

Feature representation



3rd layer
“Objects”



2nd layer
“Object parts”



1st layer
“Edges”

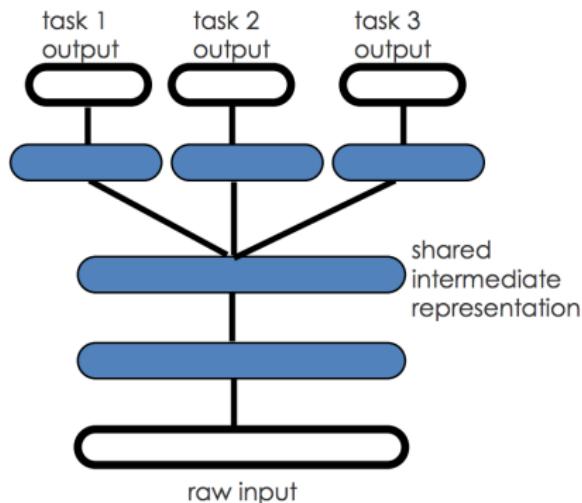


Pixels

Deep neural networks: Motivation

Shared low-level representations

- Multi-task learning
- Unsupervised training



Deep neural networks: Challenges

High memory requirements

- Memory is used to store input data, weight parameters and activations as an input propagates through the network
- Activations from a forward pass must be retained until they can be used to calculate the error gradients in the backwards pass
- Example: 50-layer neural network
 - 26 million weight parameters, 16 million activations in the forward pass
 - 168MB memory (assuming 32-bit float)

Parallelize computations with GPU (graphics processing units)

Deep neural networks: Challenges

Backpropagation does not work well

- Deep networks trained with backpropagation (without unsupervised pretraining) perform worse than shallow networks
- Gradient is progressively getting more dilute
 - Weight correction is minimal after moving back a couple of layers
- High risk of getting “stuck” to local minima
- In practice, a small portion of data is labelled

Perform pretraining to mitigate this issue

Example error rates with and without pretraining

	train.	valid.	test
DBN, unsupervised pre-training	0%	1.2%	1.2%
Deep net, auto-associator pre-training	0%	1.4%	1.4%
Deep net, supervised pre-training	0%	1.7%	2.0%
Deep net, no pre-training	.004%	2.1%	2.4%
Shallow net, no pre-training	.004%	1.8%	1.9%

(Bengio et al., NIPS 2007)

Overview

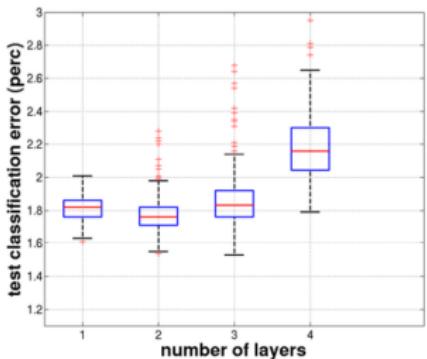
- Deep neural networks
 - Motivation & Challenges
 - Unsupervised pretraining: Deep belief networks & autoencoders
 - (Supervised) fine-tuning
 - Convolutional neural networks

Deep neural networks: Unsupervised Pretraining

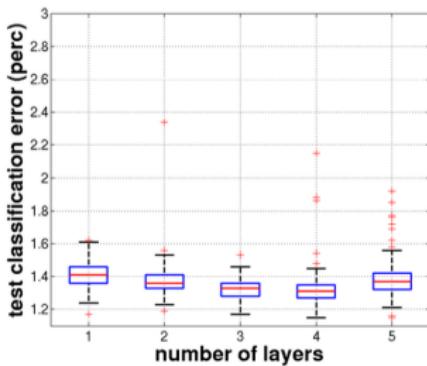
- This idea came into play when research studies found that a DNN trained on a particular task (e.g. object recognition) can be applied on another domain (e.g. object subcategorization) giving state-of-the-art results
- **1st part: Greedy layer-wise unsupervised pre-training**
 - Each layer is pre-trained with an unsupervised learning algorithm
 - Learning a nonlinear transformation that captures the main variations in its input (the output of the previous layer)
- **2nd part: Supervised fine-tuning**
 - The deep architecture is fine-tuned with respect to a supervised training criterion with gradient-based optimization
 - We will examine the **deep belief networks** and **stacked autoencoders**

Unusual form of regularization: minimizing variance and introducing bias towards configurations of the parameter space that are useful for unsupervised learning

Deep neural networks: Unsupervised Pretraining



Without pre-training



With pre-training

[Source: Erhan et al., 2010]

Deep neural networks: Unsupervised Pretraining

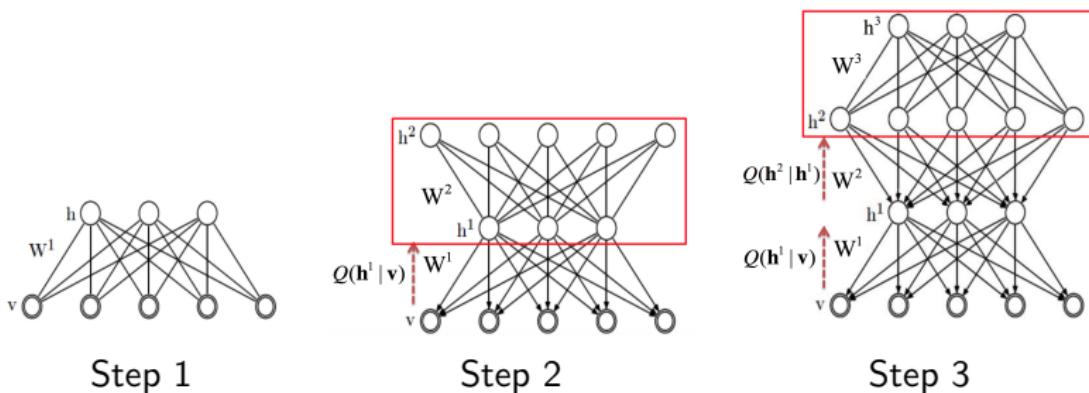
Deep Belief Networks (Hinton et al. 2006)

- Pretraining is implemented by stacking several layers of Restricted Boltzmann Machines (RBM) in a greedy manner
- Assuming joint distribution between hidden h_i and observed variables x_j with parameters $\mathbf{W}, \mathbf{b}, \mathbf{c}$
$$P(\mathbf{x}, \mathbf{h}) \propto \exp(\mathbf{h}^T \mathbf{W} \mathbf{x} + \mathbf{b}^T \mathbf{x} + \mathbf{c}^T \mathbf{h})$$
$$P(\mathbf{x}|\mathbf{h}) = \prod_j P(x_j|\mathbf{h}), P(x_j = 1|\mathbf{h}) = \text{sigmoid}(b_j + \sum_i W_{ij} h_i)$$
$$P(\mathbf{h}|\mathbf{x}) = \prod_i P(h_i|\mathbf{x}), P(h_i = 1|\mathbf{x}) = \text{sigmoid}(c_i + \sum_j W_{ij} x_j)$$
- RBM trained by approximate stochastic gradient descent
- This representation is extended to all hidden layers
- The RBM parameters correspond to the parameters of the feed-forward multi-layer neural network

Deep neural networks: Unsupervised Pretraining

Deep Belief Networks (Hinton et al. 2006)

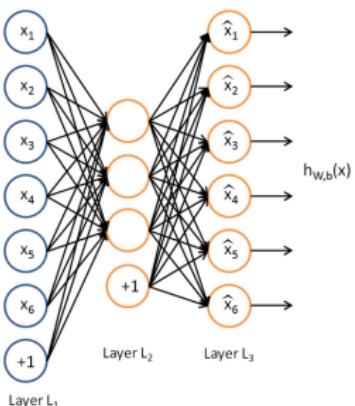
- Step 1: Construct an RBM with an input and hidden layer and train to find $\mathbf{W}^{(1)}$
- Step 2: Stack another hidden layer on top of the RBM to form a new RBM
 - Fix \mathbf{W}^1 . Assume $\mathbf{h}^{(1)}$ as input. Train to find $\mathbf{W}^{(2)}$.
- Step 3: Continue to stack layers and find weights $\mathbf{W}^{(3)}$, etc.



Deep neural networks: Unsupervised Pretraining

Autencoder

- Unsupervised algorithm that tries to learn an approximation of the identity function $h_{W,b}(x) \approx x$
- Trivial problem unless we place constraints on the network, such as by limiting the number of hidden units, we can discover interesting structure about the data
e.g. if some of the input features are correlated, then this algorithm will be able to discover some of those correlations

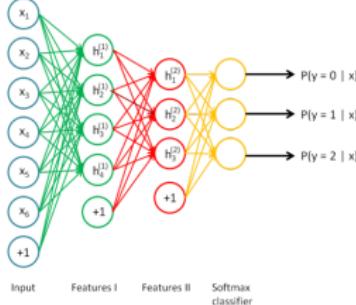
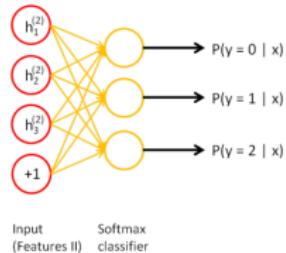
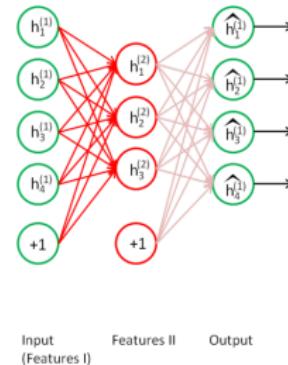
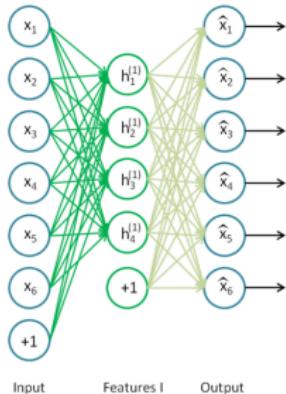


- $\alpha_j^{(i)} = f(W_{i1}^{(1)}x_1 + W_{i2}^{(1)}x_2 + \dots + b_i^{(1)})$
- Trained using back-propagation and additional sparsity constraints
- Can be also used for feature transformation

[http://ufldl.stanford.edu/wiki/index.php/Autoencoders_and_Sparsity]

Deep neural networks: Unsupervised Pretraining

Stacked Autencoders for pretraining



Deep neural networks: Unsupervised Pretraining

Stacked Autencoders for pretraining

- Capture a “hierarchical grouping” of the input
- First layer learns a good representation of input features (e.g. edges)
- Second layer learns a good representation of the patterns in the first layer (e.g. corners), etc.

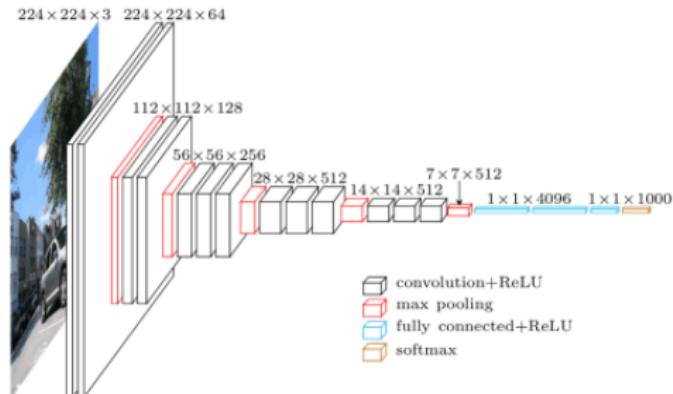
http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders

Overview

- Deep neural networks
 - Motivation & Challenges
 - Unsupervised pretraining: Deep belief networks & autoencoders
 - (Supervised) fine-tuning
 - Convolutional neural networks

Deep neural networks: Fine-tuning

- Taking advantage of labelled data from large (publicly available) datasets, e.g., VGG16
- Tweak the parameters of an already trained network so that it adapts to the new task at hand
- Initial layers → learn general features
- Last layers → learn features more specific to the task of interest
- Fine-tuning freezes the first layers, and relearns weights from the last

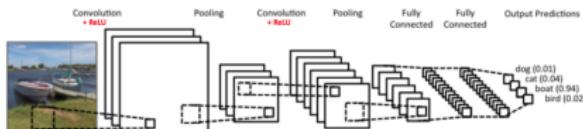


Overview

- Deep neural networks
 - Motivation & Challenges
 - Unsupervised pretraining: Deep belief networks & autoencoders
 - (Supervised) fine-tuning
 - Convolutional neural networks

Convolutional neural networks

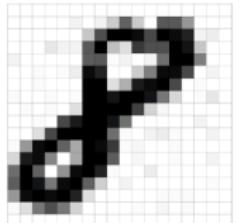
- Similar to regular neural networks
 - made up of neurons, each with an input and an activation function
 - have weights and biases to be learned
 - have a loss function on the last (fully-connected) layer
- Explicit assumption that the inputs are **images**
 - vastly reduce the amount of parameters in the network



Convolutional neural networks

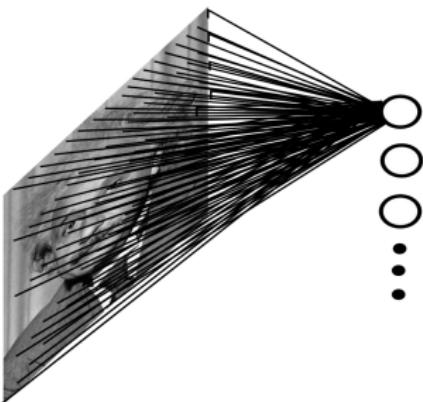
Image representation

- Grayscale image (1-channel)
 - 2d-matrix
 - each pixel ranges from 0 to 255 - 0: black, 255: white
 - Color image (3-channel, RGB)
 - three 2d-matrices stacked over each other
 - each with pixel values ranging between 0 and 255



Convolutional neural networks

A fully connected neural network with image input

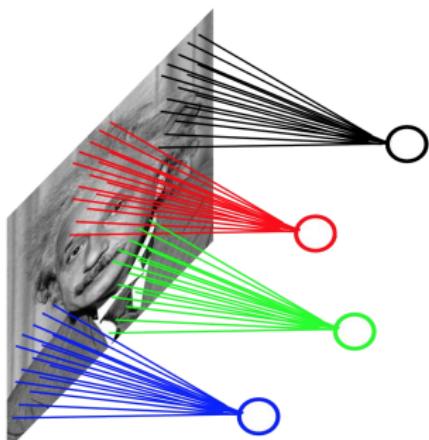


- 1000 × 1000 image, 1M hidden units
→ 10^{12} parameters
- Since spatial correlation is local, we can significantly simplify this

Convolutional neural networks

Idea 1: Convolution

A **locally** connected neural network with image input

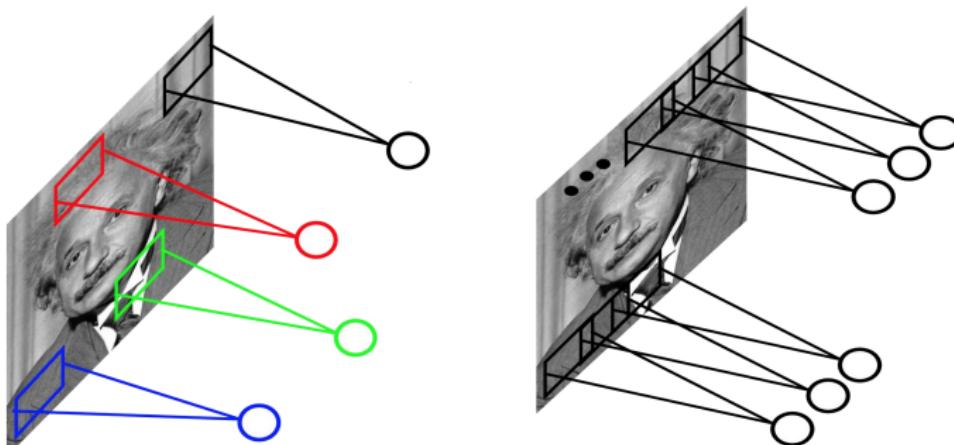


- 1000×1000 image, 1M hidden units,
 10×10 filter size $\rightarrow 10^8$ parameters
- Since spatial correlation is local, we can significantly simplify this

Convolutional neural networks

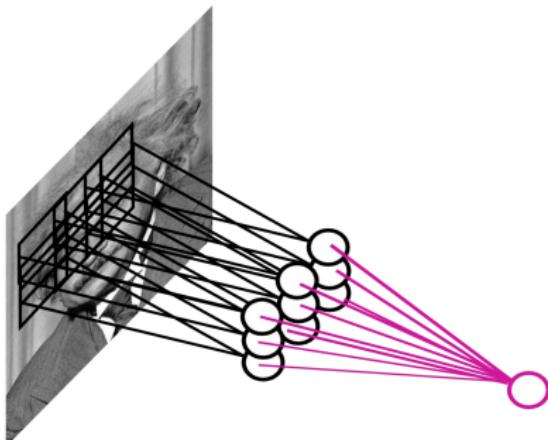
Idea 2: Weight sharing

- **Stationarity:** Statistics are similar at different locations
- Share the same parameters across different locations



Convolutional neural networks

Idea 3: Max-pooling



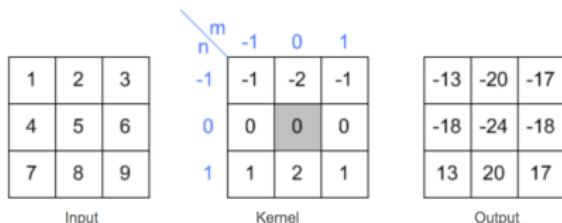
- Let us assume filter is an “eye” detector
- How can we make the detection robust to the exact location of the eye?
- By **pooling** (e.g., max or average) filter responses at different locations we gain robustness to the exact spatial location of features

Convolutional neural networks: The convolution operation

Convolution 2d-example

- Convolution is the mathematical operation that implements filtering
- Given an input image $x[m, n]$ and an impulse response $h[m, n]$ (filter or kernel), the convolution output can be written as

$$y[m, n] = x[m, n] * h[m, n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i, j]h[m - i, n - j]$$



http://www.songho.ca/dsp/convolution/convolution2d_example.html

Convolutional neural networks: The convolution operation

Convolution 2d-example

1	2	1
0	0	0
-1	1 2	
-2 4		5
7 8		9

$$\begin{aligned}
 y[0,0] &= x[-1,-1] \cdot h[1,1] + x[0,-1] \cdot h[0,1] + x[1,-1] \cdot h[-1,1] \\
 &\quad + x[-1,0] \cdot h[1,0] + x[0,0] \cdot h[0,0] + x[1,0] \cdot h[-1,0] \\
 &\quad + x[-1,1] \cdot h[1,-1] + x[0,1] \cdot h[0,-1] + x[1,1] \cdot h[-1,-1] \\
 &= 0 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 + 0 \cdot 0 + 1 \cdot 0 + 2 \cdot 0 + 0 \cdot (-1) + 4 \cdot (-2) + 5 \cdot (-1) = -13
 \end{aligned}$$

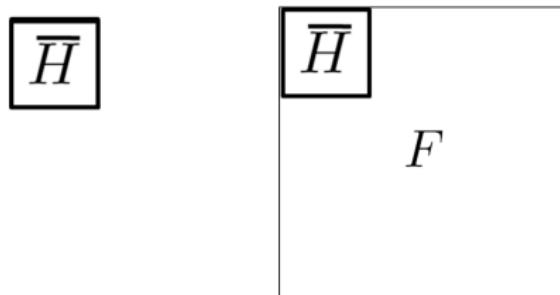
1	2	1
0	0	0
-1	1 2	
-2 4		5
7 8		9

$$\begin{aligned}
 y[1,0] &= x[0,-1] \cdot h[1,1] + x[1,-1] \cdot h[0,1] + x[2,-1] \cdot h[-1,1] \\
 &\quad + x[0,0] \cdot h[1,0] + x[1,0] \cdot h[0,0] + x[2,0] \cdot h[-1,0] \\
 &\quad + x[0,1] \cdot h[1,-1] + x[1,1] \cdot h[0,-1] + x[2,1] \cdot h[-1,-1] \\
 &= 0 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 + 1 \cdot 0 + 2 \cdot 0 + 3 \cdot 0 + 4 \cdot (-1) + 5 \cdot (-2) + 6 \cdot (-1) = -20
 \end{aligned}$$

http://www.songho.ca/dsp/convolution/convolution2d_example.html

Convolutional neural networks: The convolution operation

Convolution operation

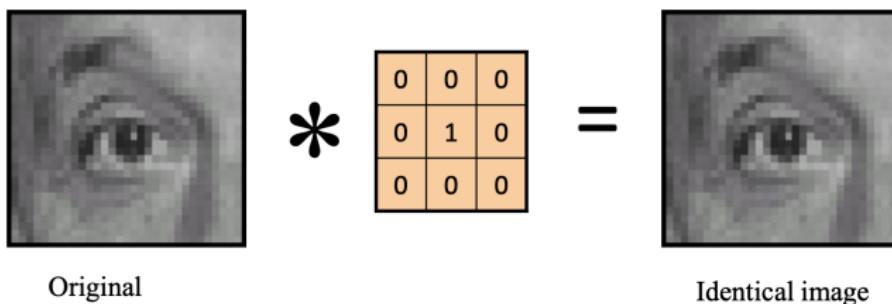


Convolutional neural networks: The convolution operation

Mean filtering

Convolutional neural networks: The convolution operation

Linear filters: Example

$$\text{Original} \quad * \quad \begin{matrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{matrix} = \text{Identical image}$$


Convolutional neural networks: The convolution operation

Linear filters: Example



*

0	0	0
1	0	0
0	0	0

=



Original

Shifted left
By 1 pixel

Convolutional neural networks: The convolution operation

Linear filters: Example

$$\text{Original} \quad * \quad \frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} = \text{Blur (with a mean filter)}$$

Convolutional neural networks: The convolution operation

Linear filters: Example



Original

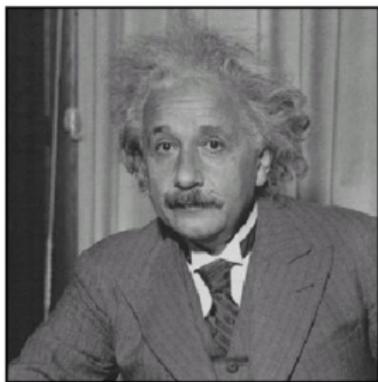
$$\text{Original} * \left(\begin{matrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{matrix} - \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \right) =$$



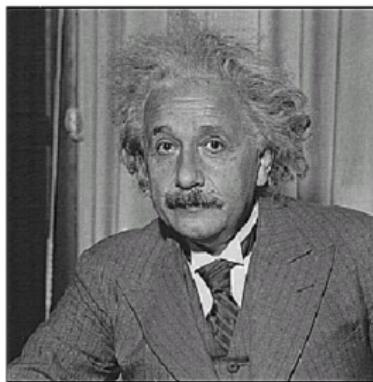
Sharpening filter
(accentuates edges)

Convolutional neural networks: The convolution operation

Linear filters: Example



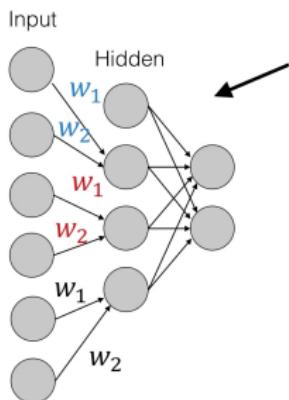
before



after

Convolutional neural networks: The convolution operation

Image 1d-convolution

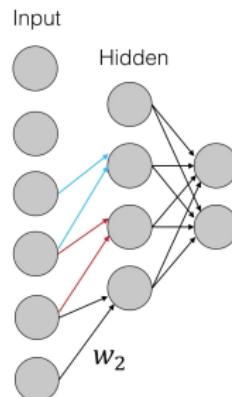


1-d convolution with

- filters: 1
- filter size: 2
- stride: 2

1-d convolution with

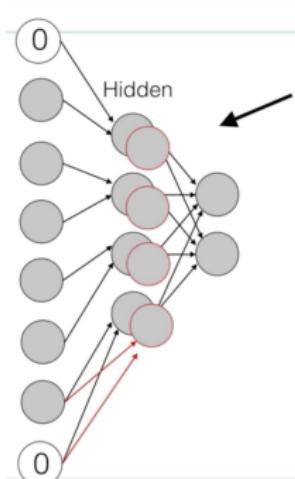
- filters: 1
- filter size: 2
- stride: **1**



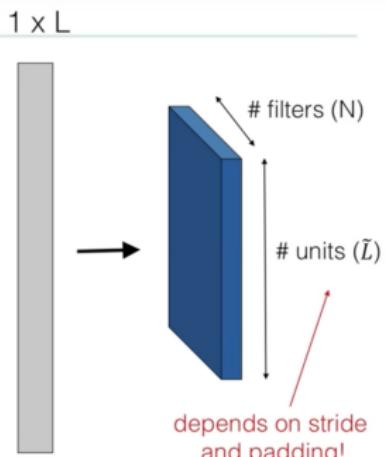
<https://www.nervanasys.com/convolutional-neural-networks/>

Convolutional neural networks: The convolution operation

Image 1d-convolution



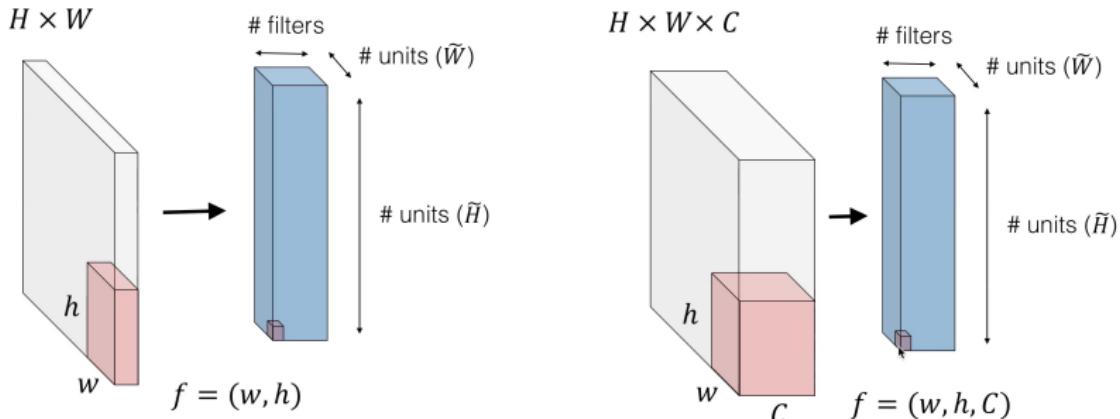
- 1-d convolution with
- filters: **2**
 - filter size: 2
 - stride: 2
 - padding: 1



<https://www.nervanasys.com/convolutional-neural-networks/>

Convolutional neural networks: The convolution operation

Image 2d-convolution



<https://www.nervanasys.com/convolutional-neural-networks/>

Also check:

<http://cs231n.github.io/assets/conv-demo/index.html>

<https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/> (figure 6)

Convolutional neural networks: The convolution operation

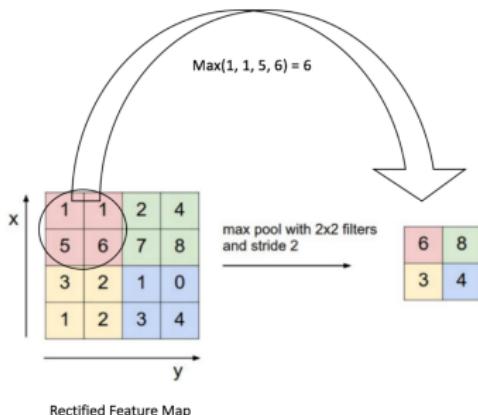
Image 2d-convolution hyperparameters

- Depth: the number of filters we use for the convolution operation
- Stride: the number of pixels by which we slide our filter matrix over the input
- Zero-padding: padding the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix

Convolutional neural networks: Max-pooling

Spatial Pooling (also called subsampling or downsampling)

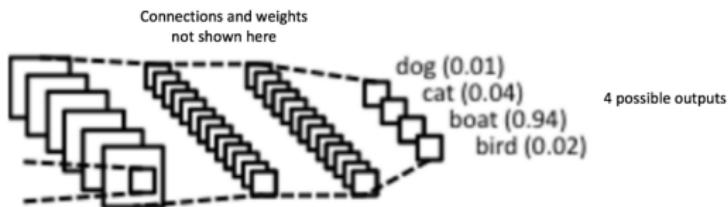
- Reduces the dimensionality of each feature map but retains the most important information
- Can be of different types: Max, Average, Sum etc.
- Makes the input representations (feature dimension) smaller and more manageable
- Promotes an almost scale invariant representation of the image



Convolutional neural networks: Final fully connected layer

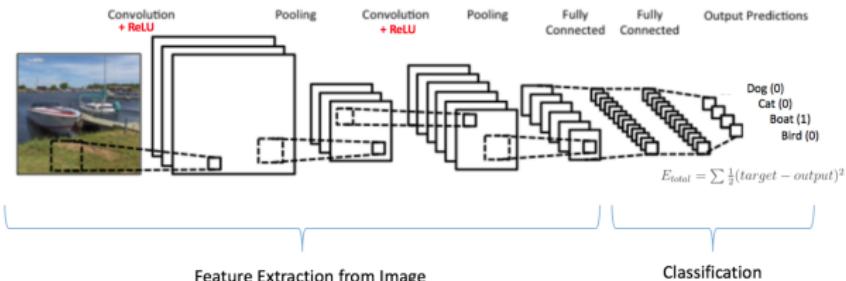
Final fully connected layer

- Traditional multilayer perceptron
- Yields the classification/regression result

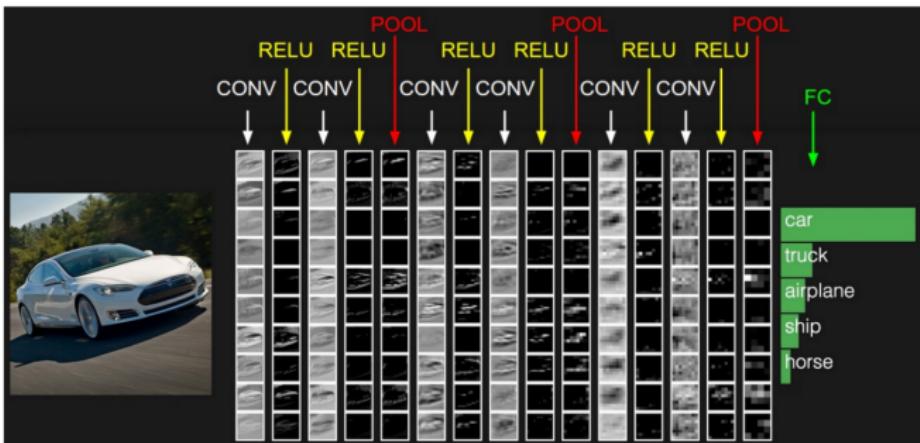


Convolutional neural networks: Putting it all together

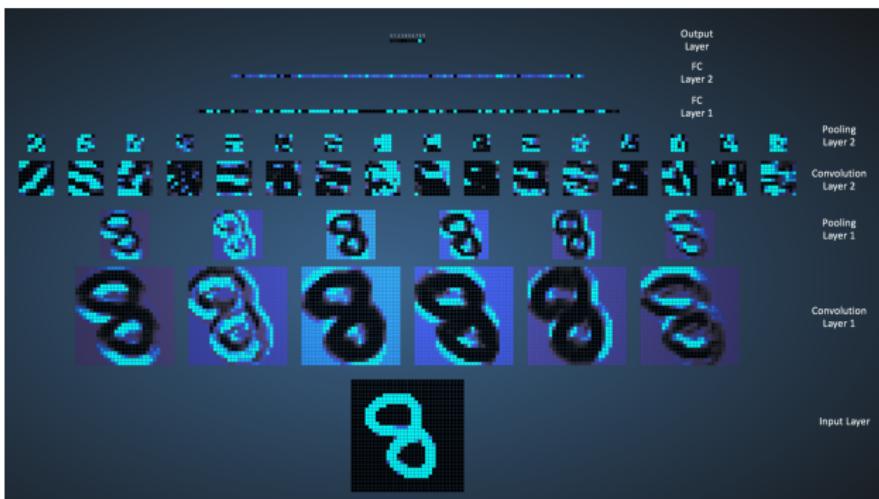
- **Step 1:** Initialize weights
- **Step 2:** Take first image as input and go through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the fully connected layer) and finds the output probabilities for each class
- **Step 3:** Calculate the total error at the output layer
- **Step 4:** Use backpropagation to update the weights, which are adjusted in proportion to their contribution to the total error
- **Step 5:** Repeat Steps 1-4 for all train images



Convolutional neural networks: Examples



Convolutional neural networks: Examples



Convolutional neural networks: Hyperparameter tuning

- Learning rate: how much to update the weight during optimization
- Number of epochs: number of times the entire training set pass through the neural network
- Batch size: the number of samples in the training set for weight update
- Activation function: the function that introduces non-linearity to the model (e.g. sigmoid, tanh, ReLU, etc.)
- Number of hidden layers and units
- Weight initialization: e.g., uniform distribution
- Dropout for regularization: probability of dropping a unit

We can perform grid or randomized search over all parameters

What have we learnt so far

- DNNs allow hierarchical representations learned from raw data
- Challenges in terms of training → pretraining
 - deep belief networks
 - autoencoders
- Convolutional neural networks → image
 - convolution: local image properties
 - weight sharing: stationarity
 - max-pooling: robustness in the representation and reduced cost